Submission Worksheet

CLICK TO GRADE

https://learn.ethereallab.app/assignment/IT114-003-F2024/it114-module-4-sockets-part-1-3/grade/vvh

Course: IT114-003-F2024
Assigment: [IT114] Module 4 Sockets Part 1-3
Student: Valeria C. (vvh)

Submissions:

Submission Selection

1 Submission [submitted] 10/7/2024 10:50:43 PM

Instructions

COLLAPSE

Overview Video: https://youtu.be/5a5HL0n6jek

1. Create a new branch for this assignment
2. If you haven't, go through the socket lessons and get each part implemented (parts 1-3)
    1. You'll probably want to put them into their own separate folders/packages (i.e., Part1, Part2, Part3) These are for your reference
3. Part 3, below, is what's necessary for this HW
    3. https://github.com/MattToegel/IT114/tree/M24-Sockets-Part3
4. Create a new folder called Part3HW (copy of Part3)
5. Make sure you have all the necessary files from Part3 copied here and fix the package references at the top of each file
    1. Add/commit/push the branch
    2. Create a pull request to main and keep it open
6. Implement **two** of the following **server-side** activities for all connected clients (majority of the logic should be processed server-side and broadcasted/sent to all clients if/when applicable)
    1. Simple number guesser where all clients can attempt to guess while the game is active
        1. Have a /start command that activates the game allowing guesses to be interpreted
        2. Have a /stop command that deactivates the game, guesses will be treated as regular messages (i.e., guess messages are ignored)
        3. Have a /guess command that include a value that is processed to see if it matches the hidden number (i.e., */guess 5*)
            1. Guess should only be considered when the game is active
            2. The response should include who guessed, what they guessed, and whether or not it was correct (i.e., Bob guessed 5 but it was not correct)
            3. No need to implement complexities like strikes

2. Coin toss command (random heads or tails)
    1. Command should be something logical like /flip or /toss or /coin or similar
    2. The result should mention *who* did *what* and got what *result* (i.e., Bob Flipped a coin and got heads)
3. Dice roller given a command and text format of "/roll #d#" (i.e., /roll 2d6)
    1. Command should be in the format of /roll #d# (i.e., /roll 1d10)
    2. The result should mention *who* did *what* and got what *result* (i.e., Bob rolled 1d10 and got 7)
4. Math game (server outputs a basic equation, first person to guess it correctly gets congratulated and a new equation is given)
    1. Have a /start command that activates the game allowing equaiton to be answered
    2. Have a /stop command that deactivates the game, answers will be treated as regular messages (i.e., any game related commands when stopped will be ignored)
    3. Have an answer command that include a value that is processed to see if it matches the hidden number (i.e., */answer 15*)
        1. The response should include who answered, what they answered, and whether or not it was correct (i.e., Bob answered 5 but it was not correct)
5. Private message (a client can send a message targetting another client where only the two can see the messages)
    1. Command can be /pm, /dm followed by the user's name or an @ preceding the users name (clearly note which)
    2. The server should properly check the target audience and send the response to the original sender and to the receiver (no one else should get the message)
    3. Alternatively (make note if you do this and show evidence) you can add support to private message multiple people at once. Evidence should show a larger number of clients than the target list of the private message to show it works. Note to grader: if this is accomplished add 0.5 to total final grade on Canvas
6. Message shuffler (randomizes the order of the characters of the given message)
    1. Command should be /shuffle or /randomize (clearly mention what you chose) followed by the message to shuffle (i.e., /shuffle hello everybody)
    2. The message should be sent to all clients showing it's from the user but randomized
        1. Example: Bob types */command* hello and everyone recevies Bob: lleho
7. Fill in the below deliverables
8. Save the submission and generated output PDF
9. Add the PDF to the Part3HW folder (local)
10. Add/commit/push your changes
11. Merge the pull request
12. Upload the same PDF to Canvas

Branch name: M4-Sockets3-Homework

Group

100%

Group: Baseline
Tasks: 1
Points: 2

COLLAPSE

Task

100%

Group: Baseline
Task #1: Demonstrate Baseline Code Working
Weight: ~100%
Points: ~2.00

COLLAPSE

Details:
This can be a single screenshot if everything fits, or can be multiple screenshots

Columns: 4

Sub-Task 100% Group: Baseline Task #1: Demonstrate Baseline Code Working

Sub-Task 100% Group: Baseline Task #1: Demonstrate Baseline Code Working

Sub-Task 100% Group: Baseline Task #1: Demonstrate Baseline Code Working

Sub-Task 100% Group: Baseline Task #1: Demonstrate Baseline Code Working

Task Screenshots

Gallery Style: 2 Columns

4 2 1

server terminal screenshot

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

Task Screenshots

Gallery Style: 2 Columns

4 2 1

client server screenshot

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's being shown

Task Screenshots

Gallery Style: 2 Columns

4 2 1

clients receiving all broadcasted messages correctly as instructed

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's

Task Screenshots

Gallery Style: 2 Columns

4 2 1

prove that parts 1-3 were added to my repo along with part3hw as instructed

Caption(s) (required) ✓
Caption Hint:
Describe/highlight what's

being shown *being shown*

End of Task 1

End of Group: Baseline
Task Status: 1/1

Group

100%

Group: Feature 1
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Feature 1
Task #1: Solution
Weight: ~100%
Points: ~3.00

COLLAPSE

Columns: 2

Sub-Task

100%

Group: Feature 1
Task #1: Solution
Sub Task #1: Show the code related to the feature (ucid and date must be present as a comment)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

coin toss implementation screenshot

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Mention specific feature and explain sufficiently and concisely the implementation (should be aligned with code snippets)
Response:

Sub-Task

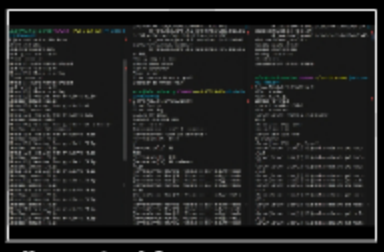
100%

Group: Feature 1
Task #1: Solution
Sub Task #2: Show the feature working (i.e., all terminals and their related output)

Task Screenshots

Gallery Style: 2 Columns

4 2 1



all terminal feature demonstration for coin toss feature

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

The coin toss code simulates a random outcome of either heads or tails using java's Math.random() method. This method generates a random double value between 0.0 and 1.0. I used Math.random() < 0.5 to divide the range in half so values less than 0.5 represent heads, and values greater than or equal to 0.5 represent tails. This approach simulates a 50/50 chance of either result. The result is then formatted into a message using String.format(), which includes the user's id and the outcome of the coin toss either heads or tails. The relay() method is then called to broadcast this result to all connected clients, ensuring that everyone sees the result of the toss.

End of Task 1

End of Group: Feature 1
Task Status: 1/1

Group

100%

Group: Feature 2
Tasks: 1
Points: 3

COLLAPSE

Task

100%

Group: Feature 2
Task #1: Solution
Weight: ~100%
Points: ~3.00

COLLAPSE

Columns: 2

Sub-Task

100%

Group: Feature 2
Task #1: Solution
Sub Task #1: Show the code related to the feature (ucid and date must be present as a comment)

Sub-Task

100%

Group: Feature 2
Task #1: Solution
Sub Task #2: Show the feature working (i.e., all terminals and their related output)

Task Screenshots

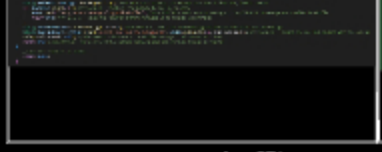
Gallery Style: 2 Columns

4 2 1

Task Screenshots

Gallery Style: 2 Columns

4 2 1



part 1 message shuffling code implementation

part2 message shuffler code implementation

message shuffler terminal demonstration screenshot

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Caption(s) (required) ✓
Caption Hint: *Describe/highlight what's being shown*

Task Response Prompt

Mention specific feature and explain sufficiently and concisely the implementation (should be aligned with code snippets)s

Response:

The message shuffling process code takes a string provided by the user, breaks it down into its individual characters, randomizes the order of those characters, and then broadcasts the shuffled result to all connected clients. It first extracts the part of the message after the /shuffle command. This extracted message is converted into a list of characters using a for loop and added to an ArrayList. The Collections.shuffle() method is then used to randomly rearrange the characters in this list. After shuffling, the characters are recombined into a new string using a StringBuilder. This shuffled string is then formatted into a message (using String.format()) that includes the user's id and the shuffled message. Finally, the relay() method is called to broadcast the newly shuffled message to all connected clients.

End of Task 1

End of Group: Feature 2
Task Status: 1/1

Group

100%

Group: Misc
Tasks: 3
Points: 2

COLLAPSE

Task

100%

Group: Misc
Task #1: Reflection
Weight: ~33%
Points: ~0.67

COLLAPSE

Sub-Task

100%

Group: Misc
Task #1: Reflection
Sub Task #1: Learn anything new? Face any challenges? How did you overcome any issues?

Task Response Prompt

Provide at least a few logical sentences

Response:

I was not expecting this assignment taking me a lot of hours. I may have chosen the easiest ones, but I struggle a lot at the beginning at least with the first coin toss first because I get to run the server and get the clients connected to it, but when I type the /flip it was not working and it was not displaying to the other clients. then when I tried to implement the second feature I chose for the message shuffler, when I wrote the /shuffle it was not shuffling the message and the problem was than when I was writing the code, I save it, but I didnt compile it again and there was an error with the imports because I actually told me was missing two, and then after adding those two and compiling it actually run. It was something so simple, but essential for the program execution.

End of Task 1

Task

100%

Group: Misc
Task #2: Pull request link
Weight: ~33%
Points: ~0.67

COLLAPSE

Details:
URL should end with /pull/# and be related to this assignment

Task URLs

URL #1
https://github.com/vvh24/vvh-IT114-003/pull/9

URL
https://github.com/vvh24/vvh-IT114-003/pull/9

End of Task 2

Task

100%

Group: Misc
Task #3: Waka Time (or related) Screenshot
Weight: ~33%
Points: ~0.67

COLLAPSE

Details:

Screenshot clearly shows what files/project were being worked on (the duration of time doesn't correlated with the grade for this item)

Task Screenshots

Gallery Style: 2 Columns

4 2 1

vscode wakatime demonstration screenshot

wakatime repo time screenshot

Files

3 hrs 33 mins	Module4/Part3HW/Server.java
25 mins	...4/Part3HW/ServerThread.java
21 mins	Module4/Part3HW/Client.java
4 mins	Module4/Part1/Server.java
2 mins	Module4/Part1/Client.java
1 min	...le4/Part3/ServerThread.java
1 min	Module4/Part3/Client.java
52 secs	Module4/Part2/Server.java
25 secs	Module4/Part2/Client.java
16 secs	Module4/Part3/Server.java
0 secs	Module3/NumberGuesser4.java

files wakatime screenshot

End of Task 3

End of Group: Misc
Task Status: 3/3

End of Assignment